

TECH SHOP

Implement OOPs

Task 1: Classes and Their Attributes:

Task 2: Class Creation:

- Create the classes (Customers, Products, Orders, OrderDetails and Inventory) with the specified attributes.
- Implement the constructor for each class to initialize its attributes.
- Implement methods as specified.

Task 3: Encapsulation:

- Implement encapsulation by making the attributes private and providing public properties (getters and setters) for each attribute.
- Add data validation logic to setter methods (e.g., ensure that prices are non-negative, quantities are positive integers).

Customers Class:

Attributes:

- CustomerID (int)
- FirstName (string)
- LastName (string)
- Email (string)
- Phone (string)
- Address (string)

Methods:

- CalculateTotalOrders(): Calculates the total number of orders placed by this customer.
- GetCustomerDetails(): Retrieves and displays detailed information about the customer.
- UpdateCustomerInfo(): Allows the customer to update their information (e.g., email, phone, or address).

```
InventoryDao.py  OrderDetailsDao.py  OrderDao.py  Customer.py X
TechShop > entity > Customer.py > Customer > update_customer_info
1  import re
2  from exception.exceptions import InvalidDataException
3
4
5  class Customer:
6      def __init__(self, customer_id=None, first_name="", last_name="", email="", phone="", address=""):
7          self.__customer_id = customer_id
8          self.__first_name = first_name
9          self.__last_name = last_name
10         self.__email = self.__validate_email(email)
11         self.__phone = phone
12         self.__address = address
13         self.orders = []
14
15     def __validate_email(self, email):
16         if re.match(r"^[^@]+@[^@]+\.[^@]+$", email):
17             return email
18         else:
19             raise InvalidDataException("Invalid email address provided.")
20
21     def calculate_total_orders(self):
22         return len(self.orders)
23
24     def get_customer_details(self):
25         return f"Customer ID: {self.__customer_id}\nName: {self.__first_name} {self.__last_name}\nEmail:
26
27     def update_customer_info(self, email=None, phone=None, address=None):
28         if email:
29             self.email = email
30         if phone:
31             self.phone = phone
32         if address:
33             self.address = address
34
35     @property
36     def customer_id(self):
37         return self.__customer_id
38
```

techShop > entity > Customer.py > Customer > update_customer_info

```
5 class Customer:
38
39     @property
40     def first_name(self):
41         return self.__first_name
42
43     @first_name.setter
44     def first_name(self, first_name):
45         if isinstance(first_name, str):
46             self.__first_name = first_name
47         else:
48             raise InvalidDataException("First name must be a string.")
49
50     @property
51     def last_name(self):
52         return self.__last_name
53
54     @last_name.setter
55     def last_name(self, last_name):
56         if isinstance(last_name, str):
57             self.__last_name = last_name
58         else:
59             raise InvalidDataException("Last name must be a string.")
60
61     @property
62     def email(self):
63         return self.__email
64
65     @email.setter
66     def email(self, email):
67         self.__email = self.__validate_email(email)
68
69     @property
70     def phone(self):
71         return self.__phone
72
73     @phone.setter
74     def phone(self, phone):
75         # Phone number validation: must be a string containing only digits
76         if isinstance(phone, str) and phone.isdigit():
77             self.__phone = phone
78         else:
79             raise InvalidDataException("Phone number must be a string containing only digits.")
80
81     @property
82     def address(self):
83         return self.__address
84
```

```
84
85     @address.setter
86     def address(self, address):
87         if isinstance(address, str):
88             self.__address = address
89         else:
90             raise InvalidDataException("Address must be a string.")
91
92
93
94
```

Products Class:

Attributes:

- ProductID (int)
- ProductName (string)
- Description (string)
- Price (decimal)

Methods:

- GetProductDetails(): Retrieves and displays detailed information about the product.
- UpdateProductInfo(): Allows updates to product details (e.g., price, description).
- IsProductInStock(): Checks if the product is currently in stock.

```
TechShop > entity > Product.py > Product > product_name
1 class Product:
2     def __init__(self, product_id=None, product_name="", description="", price=0.00, in_stock=True):
3         self.__product_id = product_id
4         self.__product_name = product_name
5         self.__description = description
6         self.__price = price
7         self.__in_stock = in_stock
8
9     def get_product_details(self):
10        return f"Product ID: {self.__product_id}\nName: {self.__product_name}\nDescription: {self.__description}\nPrice: ${self.__price}\nIn Sto
11
12    def update_product_info(self, price=None, description=None, in_stock=None):
13        if price is not None:
14            self.__price = price
15        if description is not None:
16            self.__description = description
17        if in_stock is not None:
18            self.__in_stock = in_stock
19
20    def is_product_in_stock(self):
21        return self.__in_stock
22
23    @property
24    def product_id(self):
25        return self.__product_id
26
27    @property
28    def product_name(self):
29        return self.__product_name
30
31    @product_name.setter
32    def product_name(self, value):
33        if isinstance(value, str):
34            self.__product_name = value
35        else:
36            raise ValueError("Product name must be a string.")
37
38    @property
39    def description(self):
40        return self.__description
41
42    @description.setter
43    def description(self, value):
44        if isinstance(value, str):
45            self.__description = value
46        else:
47            raise ValueError("Description must be a string.")
48
```

```

3
9     @property
0     def price(self):
1         return self.__price
2
3     @price.setter
4     def price(self, value):
5         if isinstance(value, (int, float)):
6             if value >= 0: # Check if price is non-negative
7                 self.__price = value
8             else:
9                 raise ValueError("Price must not be negative.")
0         else:
1             raise ValueError("Price must be a numeric value.")
2

```

Orders Class:

Attributes:

- OrderID (int)
- Customer (Customer) - Use composition to reference the Customer who placed the order.
- OrderDate (DateTime)
- TotalAmount (decimal)

Methods:

- CalculateTotalAmount() - Calculate the total amount of the order.
- GetOrderDetails(): Retrieves and displays the details of the order (e.g., product list and quantities).
- UpdateOrderStatus(): Allows updating the status of the order (e.g., processing, shipped).
- CancelOrder(): Cancels the order and adjusts stock levels for products.

```
InventoryDao.py  OrderDetailsDao.py  OrderDao.py  Customer.py  Order.py
TechShop > entity > Order.py > Order > get_order_details
1  from datetime import datetime
2
3  from entity.Customer import Customer
4
5  class Order:
6      def __init__(self, order_id=None, customer=None, totalAmount=0, products=None):
7          self.__order_id = order_id
8          self.__customer = customer
9          self.__order_date = datetime.now()
10         self.__total_amount = totalAmount
11         self.__products = products if products else []
12
13     def calculate_total_amount(self):
14         self.__total_amount = sum(product.price for product in self.__products)
15         return self.__total_amount
16
17     def get_order_details(self):
18         pass
19
20     def update_order_status(self, status):\
21         pass
22
23     def cancel_order(self):
24         pass
25
26     @property
27     def order_id(self):
28         return self.__order_id
29
30     @property
31     def customer(self):
32         return self.__customer
33
34     @customer.setter
35     def customer(self, customer):
36         self.__customer = customer
37
38     @property
39     def order_date(self):
40         return self.__order_date
41
42     @order_date.setter
43     def order_date(self, order_date):
44         if isinstance(order_date, datetime):
45             self.__order_date = order_date
46         else:
47             raise ValueError("Order date must be a datetime object.")
```

```

47         raise ValueError("Order date must be a datetime object. ")
48
49     @property
50     def total_amount(self):
51         return self.__total_amount
52
53     @total_amount.setter
54     def total_amount(self, total_amount):
55         if isinstance(total_amount, (int, float)):
56             self.__total_amount = total_amount
57         else:
58             raise ValueError("Total amount must be a numeric value.")
59

```

OrderDetails Class:

Attributes:

- OrderDetailID (int)
- Order (Order) - Use composition to reference the Order to which this detail belongs.
- Product (Product) - Use composition to reference the Product included in the order detail.
- Quantity (int)

Methods:

- CalculateSubtotal() - Calculate the subtotal for this order detail.
- GetOrderDetailInfo(): Retrieves and displays information about this order detail.
- UpdateQuantity(): Allows updating the quantity of the product in this order detail.
- AddDiscount(): Applies a discount to this order detail.

TechShop > entity > OrderDetails.py > OrderDetails

```
1
2 from entity.Order import Order
3 from exception.exceptions import InvalidDataException
4
5
6 class OrderDetails:
7     def __init__(self, order_detail_id=None, order=None, product=None, quantity=0):
8         self.__order_detail_id = order_detail_id
9         self.__order = order
10        self.__product = product
11        self.__quantity = quantity
12
13    def calculate_subtotal(self):
14        pass
15
16    def get_order_detail_info(self):
17        pass
18
19    def update_quantity(self, new_quantity):
20        pass
21
22    def add_discount(self, discount):
23        pass
24
25    @property
26    def order_detail_id(self):
27        return self.__order_detail_id
28
29    @property
30    def order(self):
31        return self.__order
32
33    @order.setter
34    def order(self, order):
35        self.__order = order
36
37    @property
38    def product(self):
39        return self.__product
40
41    @product.setter
42    def product(self, product):
43        self.__product = product
44
45    @property
46    def quantity(self):
47        return self.__quantity
48
```



```

48
49     @quantity.setter
50     def quantity(self, quantity):
51         if isinstance(quantity, int) and quantity >= 0:
52             self.__quantity = quantity
53         else:
54             raise InvalidDataException("Quantity must be a non-negative integer.")
55

```

Inventory class:

Attributes:

- InventoryID(int)
- Product (Composition): The product associated with the inventory item.
- QuantityInStock: The quantity of the product currently in stock.
- LastStockUpdate

Methods:

- GetProduct(): A method to retrieve the product associated with this inventory item.
- GetQuantityInStock(): A method to get the current quantity of the product in stock.
- AddToInventory(int quantity): A method to add a specified quantity of the product to the inventory.
- RemoveFromInventory(int quantity): A method to remove a specified quantity of the product from the inventory.
- UpdateStockQuantity(int newQuantity): A method to update the stock quantity to a new value.
- IsProductAvailable(int quantityToCheck): A method to check if a specified quantity of the product is available in the inventory.
- GetInventoryValue(): A method to calculate the total value of the products in the inventory based on their prices and quantities.
- ListLowStockProducts(int threshold): A method to list products with quantities below a specified threshold, indicating low stock.
- ListOutOfStockProducts(): A method to list products that are out of stock.
- ListAllProducts(): A method to list all products in the inventory, along with their quantities.

```
TechShop > entity > Inventory.py > Inventory
1  from datetime import datetime
2  from exception.exceptions import InsufficientStockException, InvalidDataException
3
4  class Inventory:
5      def __init__(self, inventory_id=None, product=None, quantity_in_stock=0, last_stock_update=None):
6          self.__inventory_id = inventory_id
7          self.__product = product
8          self.__quantity_in_stock = quantity_in_stock if quantity_in_stock > 0 else 0
9          self.__last_stock_update = last_stock_update if last_stock_update else datetime.now()
10
11     def get_product(self):
12         return self.__product
13
14     def get_quantity_in_stock(self):
15         return self.__quantity_in_stock
16
17     def add_to_inventory(self, quantity):
18         self.__quantity_in_stock += quantity
19         self.__last_stock_update = datetime.now()
20
21     def remove_from_inventory(self, quantity):
22         if self.__quantity_in_stock >= quantity:
23             self.__quantity_in_stock -= quantity
24             self.__last_stock_update = datetime.now()
25         else:
26             raise InsufficientStockException("Insufficient quantity in stock.")
27
28     def update_stock_quantity(self, new_quantity):
29         self.__quantity_in_stock = new_quantity
30         self.__last_stock_update = datetime.now()
31
32     def is_product_available(self, quantity_to_check):
33         return self.__quantity_in_stock >= quantity_to_check
34
35     def get_inventory_value(self):
36         return self.__product.price * self.__quantity_in_stock
37
38     def list_low_stock_products(self, threshold):
39         if self.__quantity_in_stock < threshold:
40             return self.__product
41
42     def list_out_of_stock_products(self):
43         if self.__quantity_in_stock == 0:
44             return self.__product
45
46     def list_all_products(self):
47         return self.__product
48
```

```

4  class Inventory:
49
50      @property
51      def inventory_id(self):
52          return self.__inventory_id
53
54      @property
55      def product(self):
56          return self.__product
57
58      @product.setter
59      def product(self, product):
60          # Add validation logic if needed
61          self.__product = product
62
63      @property
64      def quantity_in_stock(self):
65          return self.__quantity_in_stock
66
67      @quantity_in_stock.setter
68      def quantity_in_stock(self, quantity_in_stock):
69          if isinstance(quantity_in_stock, int) and quantity_in_stock >= 0:
70              self.__quantity_in_stock = quantity_in_stock
71          else:
72              raise InvalidDataException("Quantity in stock must be a non-negative integer.")
73
74      @property
75      def last_stock_update(self):
76          return self.__last_stock_update
77
78      @last_stock_update.setter
79      def last_stock_update(self, last_stock_update):
80          if isinstance(last_stock_update, datetime):
81              self.__last_stock_update = last_stock_update
82          else:
83              raise InvalidDataException("Last stock update must be a datetime object.")

```

Task 4: Composition:

Ensure that the Order and OrderDetail classes correctly use composition to reference Customer and Product objects.

- Orders Class with Composition:

- o In the Orders class, we want to establish a composition relationship with the Customers class, indicating that each order is associated with a specific customer.

- o In the Orders class, we've added a private attribute customer of type Customers, establishing a composition relationship. The Customer property provides access to the Customers object associated with the order.

```

2
3 from entity.Customer import Customer
4
5 class Order:
6     def __init__(self, order_id=None, customer: Customer=None, totalAmount=0, products=None):
7         self.__order_id = order_id
8         self.__customer = customer
9         self.__order_date = datetime.now()
10        self.__total_amount = totalAmount
11        self.__products = products if products else []
12

```

- OrderDetails Class with Composition:

- o Similarly, in the OrderDetails class, we want to establish composition relationships with both the Orders and Products classes to represent the details of each order, including the product being ordered.

- o In the OrderDetails class, we've added two private attributes, order and product, of types Orders and Products, respectively, establishing composition relationships. The Order property provides access to the Orders object associated with the order detail, and the Product property provides access to the Products object representing the product in the order detail.

```

1
2 from entity.Order import Order
3 from entity.Product import Product
4 from exception.exceptions import InvalidDataException
5
6
7 class OrderDetails:
8     def __init__(self, order_detail_id=None, order: Order=None, product: Product=None, quantity=0):
9         self.__order_detail_id = order_detail_id
10        self.__order = order
11        self.__product = product
12        self.__quantity = quantity
13

```

- Customers and Products Classes:

- o The Customers and Products classes themselves may not have direct composition relationships with other classes in this scenario. However, they serve as the basis for composition relationships in the Orders and OrderDetails classes, respectively.

- Inventory Class:

- o The Inventory class represents the inventory of products available for sale. It can have composition relationships with the Products class to indicate which products are in the inventory.

```

TechShop > entity > Inventory.py > Inventory > get_product
1  from datetime import datetime
2  from entity.Product import Product
3  from exception.exceptions import InsufficientStockException, InvalidDataException
4
5  class Inventory:
6      def __init__(self, inventory_id=None, product: Product=None, quantity_in_stock=0, last_stock_update=None):
7          self.__inventory_id = inventory_id
8          self.__product = product
9          self.__quantity_in_stock = quantity_in_stock if quantity_in_stock>0 else 0
10         self.__last_stock_update = last_stock_update if last_stock_update else datetime.now()
11

```

Task 5: Exceptions handling

- Data Validation:

- o Challenge: Validate user inputs and data from external sources (e.g., user registration, order placement).

- o Scenario: When a user enters an invalid email address during registration.

- o Exception Handling: Throw a custom InvalidDataException with a clear error message.

- Inventory Management:

- o Challenge: Handling inventory-related issues, such as selling more products than are in stock.

- o Scenario: When processing an order with a quantity that exceeds the available stock.

- o Exception Handling: Throw an InsufficientStockException and update the order status accordingly.

- Order Processing:

- o Challenge: Ensuring the order details are consistent and complete before processing.

- o Scenario: When an order detail lacks a product reference.

- o Exception Handling: Throw an IncompleteOrderException with a message explaining the issue.

- Payment Processing:

- o Challenge: Handling payment failures or declined transactions.

- o Scenario: When processing a payment for an order and the payment is declined.

- o Exception Handling: Handle payment-specific exceptions (e.g., PaymentFailedException) and initiate retry or cancellation processes.

- File I/O (e.g., Logging):

- o Challenge: Logging errors and events to files or databases.

- o Scenario: When an error occurs during data persistence (e.g., writing a log entry).

- o Exception Handling: Handle file I/O exceptions (e.g., `IOException`) and log them appropriately.

- Database Access:

- o Challenge: Managing database connections and queries.

- o Scenario: When executing a SQL query and the database is offline.

- o Exception Handling: Handle database-specific exceptions (e.g., `SQLException`) and implement connection retries or failover mechanisms.

- Concurrency Control:

- o Challenge: Preventing data corruption in multi-user scenarios.

- o Scenario: When two users simultaneously attempt to update the same order.

- o Exception Handling: Implement optimistic concurrency control and handle `ConcurrencyException` by notifying users to retry.

- Security and Authentication:

- o Challenge: Ensuring secure access and handling unauthorized access attempts.

- o Scenario: When a user tries to access sensitive information without proper authentication.

- o Exception Handling: Implement custom `AuthenticationException` and `AuthorizationException` to handle security-related issues.

```
TechShop > exception > exceptions.py > ...
1  # exceptions.py
2
3  class InsufficientStockException(Exception):
4      def __init__(self, message="Insufficient stock."):
5          self.message = message
6          super().__init__(self.message)
7
8
9  class InvalidDataException(Exception):
10     def __init__(self, message="Invalid data provided."):
11         self.message = message
12         super().__init__(self.message)
13
14     class IncompleteOrderException(Exception):
15         def __init__(self, message="Incomplete order details."):
16             self.message = message
17             super().__init__(self.message)
18
19
20     class PaymentFailedException(Exception):
21         def __init__(self, message="Payment failed."):
22             self.message = message
23             super().__init__(self.message)
24
```

```

26
27 #product management
28 class DuplicateProductException(Exception):
29     def __init__(self, message="Product already exists."):
30         self.message = message
31         super().__init__(self.message)
32
33 class NotFoundException(Exception):
34     def __init__(self, message="Product not found."):
35         self.message = message
36         super().__init__(self.message)
37
38 class ProductHasOrdersException(Exception):
39     def __init__(self, message="Product has existing orders and cannot be removed."):
40         self.message = message
41         super().__init__(self.message)
42
43 class FileIOException(Exception):
44     def __init__(self, message="Error accessing or performing I/O operation on file."):
45         self.message = message
46         super().__init__(self.message)
47
48
49 class DatabaseOfflineException(Exception):
50     def __init__(self, message="Error executing SQL query due to database offline."):
51         self.message = message
52         super().__init__(self.message)
53
54 class ConcurrencyException(Exception):
55     def __init__(self, message="Concurrency issue occurred. Please retry."):
56         self.message = message
57         super().__init__(self.message)
58
59 class AuthenticationException(Exception):
60     def __init__(self, message="Authentication failed. Invalid username or password."):
61         self.message = message
62         super().__init__(self.message)
63
64 class AuthorizationException(Exception):
65     def __init__(self, message="Unauthorized access attempt."):
66         self.message = message
67         super().__init__(self.message)
68

```

Task 6: Collections

- Managing Products List:
 - o Challenge: Maintaining a list of products available for sale (List<Products>).
 - o Scenario: Adding, updating, and removing products from the list.
 - o Solution: Implement methods to add, update, and remove products. Handle exceptions for duplicate products, invalid updates, or removal of products with existing orders.
- Managing Orders List:
 - o Challenge: Maintaining a list of customer orders (List<Orders>).
 - o Scenario: Adding new orders, updating order statuses, and removing canceled orders.
 - o Solution: Implement methods to add new orders, update order statuses, and remove canceled orders. Ensure that updates are synchronized with inventory and payment records.
- Sorting Orders by Date:
 - o Challenge: Sorting orders by order date in ascending or descending order.

- o Scenario: Retrieving and displaying orders based on specific date ranges.
- o Solution: Use the `List<Orders>` collection and provide custom sorting methods for order date. Consider implementing `SortedList` if you need frequent sorting operations.
- Inventory Management with `SortedList`:
 - o Challenge: Managing product inventory with a `SortedList` based on product IDs.
 - o Scenario: Tracking the quantity in stock for each product and quickly retrieving inventory information.
 - o Solution: Implement a `SortedList<int, Inventory>` where keys are product IDs. Ensure that inventory updates are synchronized with product additions and removals.
- Handling Inventory Updates:
 - o Challenge: Ensuring that inventory is updated correctly when processing orders.
 - o Scenario: Decrementing product quantities in stock when orders are placed.
 - o Solution: Implement a method to update inventory quantities when orders are processed. Handle exceptions for insufficient stock.
- Product Search and Retrieval:
 - o Challenge: Implementing a search functionality to find products based on various criteria (e.g., name, category).
 - o Scenario: Allowing customers to search for products.
 - o Solution: Implement custom search methods using LINQ queries on the `List<Products>` collection. Handle exceptions for invalid search criteria.
- Duplicate Product Handling:
 - o Challenge: Preventing duplicate products from being added to the list.
 - o Scenario: When a product with the same name or SKU is added.
 - o Solution: Implement logic to check for duplicates before adding a product to the list. Raise exceptions or return error messages for duplicates.
- Payment Records List:
 - o Challenge: Managing a list of payment records for orders (`List<PaymentClass>`).
 - o Scenario: Recording and updating payment information for each order.
 - o Solution: Implement methods to record payments, update payment statuses, and handle payment errors. Ensure that payment records are consistent with order records.
- OrderDetails and Products Relationship:

- o Challenge: Managing the relationship between OrderDetails and Products.
- o Scenario: Ensuring that order details accurately reflect the products available in the inventory.
- o Solution: Implement methods to validate product availability in the inventory before adding order details. Handle exceptions for unavailable products.

```
TechShop > controller > inventoryManagement.py > InventoryManager > _find_index_by_product_id
1  from sortedcontainers import SortedList
2  from datetime import datetime
3
4
5  class InventoryManager:
6      def __init__(self):
7          self.inventory_list = SortedList()
8
9      def add_inventory(self, inventory):
10         self.inventory_list.add((inventory.product.product_id, inventory))
11
12     def remove_inventory(self, product_id):
13         index = self._find_index_by_product_id(product_id)
14         if index is not None:
15             del self.inventory_list[index]
16
17     def update_inventory_quantity(self, product_id, new_quantity):
18         index = self._find_index_by_product_id(product_id)
19         if index is not None:
20             self.inventory_list[index][1].quantity_in_stock = new_quantity
21
22     def get_inventory_by_product_id(self, product_id):
23         index = self._find_index_by_product_id(product_id)
24         if index is not None:
25             return self.inventory_list[index][1]
26         return None
27
28     def _find_index_by_product_id(self, product_id):
29         for i, (key, inventory) in enumerate(self.inventory_list):
30             if key == product_id:
31                 return i
32         return None
33
```

TechShop > controller > orderManagement.py > OrderManager > update_order_status

```
1  from exception.exceptions import NotFoundException
2
3
4  class OrderManager:
5      def __init__(self):
6          self.__orders = []
7
8      def add_order(self, order):
9          self.__orders.append(order)
10
11      def update_order_status(self, order_id, new_status):
12          for order in self.__orders:
13              if order.order_id == order_id:
14                  order.update_order_status(new_status)
15                  return
16          raise NotFoundException("Order not found in the list.")
17
18      def remove_canceled_orders(self):
19          self.__orders = [order for order in self.__orders if order.status != "canceled"]
20
21      def sort_orders_by_date(self, ascending=True):
22          self.__orders.sort(key=lambda order: order.order_date, reverse=not ascending)
23
24      def get_orders_in_date_range(self, start_date, end_date):
25          return [order for order in self.__orders if start_date <= order.order_date <= end_date]
26
27      def list_orders(self):
28          return self.__orders
29
```

TechShop > controller > productManagement.py > ProductManager > list_products

```
1  from exception.exceptions import DuplicateProductException, ProductHasOrdersException, NotFoundException
2
3
4  class ProductManager:
5      def __init__(self):
6          self.__products = []
7
8      def add_product(self, product):
9          if product not in self.__products:
10             self.__products.append(product)
11          else:
12             raise DuplicateProductException("Product already exists in the list.")
13
14      def update_product(self, product_id, new_product):
15          for i, product in enumerate(self.__products):
16              if product.product_id == product_id:
17                  self.__products[i] = new_product
18                  return
19          raise NotFoundException("Product not found in the list.")
20
21      def remove_product(self, product_id):
22          for i, product in enumerate(self.__products):
23              if product.product_id == product_id:
24                  # Check if the product has existing orders
25                  if product.has_orders():
26                      raise ProductHasOrdersException("Product cannot be removed as it has existing orders.")
27                  del self.__products[i]
28                  return
29          raise NotFoundException("Product not found in the list.")
30
31      def list_products(self):
32          return self.__products
33
```

```

TechShop > controller > paymentManagement.py > PaymentRecordsList > record_payment
1  class PaymentRecord:
2      def __init__(self, order_id, amount, status="Pending"):
3          self.order_id = order_id
4          self.amount = amount
5          self.status = status
6
7  class PaymentRecordsList:
8      def __init__(self):
9          self.payment_records = []
10
11     def record_payment(self, payment_record):
12         """
13         Record a payment for an order.
14         """
15         self.payment_records.append(payment_record)
16
17     def update_payment_status(self, order_id, new_status):
18         """
19         Update payment status for a specific order.
20         """
21         for payment_record in self.payment_records:
22             if payment_record.order_id == order_id:
23                 payment_record.status = new_status
24                 return
25         # If order_id not found
26         raise ValueError(f"No payment record found for order ID: {order_id}")
27
28     def handle_payment_error(self, order_id, error_message):
29         """
30         Handle payment errors for a specific order.
31         """
32         for payment_record in self.payment_records:
33             if payment_record.order_id == order_id:
34                 payment_record.status = "Error"
35                 payment_record.error_message = error_message
36                 return
37         # If order_id not found
38         raise ValueError(f"No payment record found for order ID: {order_id}")
39

```

Task 7: Database Connectivity

- Implement a DatabaseConnector class responsible for establishing a connection to the "TechShopDB" database. This class should include methods for opening, closing, and managing database connections.

```

TechShop > util > DBConnUtil.py > DBConnUtil
1  import mysql.connector
2  from mysql.connector import Error
3  import time
4
5  from util.DBPropertyUtil import DBPropertyUtil
6
7  class DBConnUtil:
8      __connection = None
9
10     @staticmethod
11     def getConnection():
12         property_file = r"C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop\db.properties"
13         connection_string = DBPropertyUtil().get_connection_string(property_file)
14         try:
15             DBConnUtil.__connection = mysql.connector.connect(**connection_string)
16             print("Connected to MySQL database successfully.")
17             return DBConnUtil.__connection
18         except mysql.connector.Error as err:
19             print("Error connecting to MySQL:", err)
20             return None
21
22
23     @staticmethod
24     def close_connection():
25         if DBConnUtil.__connection:
26             DBConnUtil.__connection.close()
27             print("Connection closed.")
28             DBConnUtil.__connection = None
29

```

```

TechShop > util > DBPropertyUtil.py > DBPropertyUtil > get_connection_string
1  # util/DBPropertyUtil.py
2  import configparser
3
4  class DBPropertyUtil:
5      @staticmethod
6      def get_connection_string(property_file):
7          config = configparser.ConfigParser()
8          config.read(property_file)
9          connection_string = {
10              'user': config['DATABASE']['user'],
11              'host': config['DATABASE']['host'],
12              'port': config['DATABASE'].getint('port'),
13              'passwd': config['DATABASE']['passwd'],
14              'database': config['DATABASE']['database']
15          }
16          return connection_string
17

```

- Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors, and methods for CRUD (Create, Read, Update, Delete) operations.

```

6
7 from dao.CustomerDao import CustomerDao
8 from dao.InventoryDao import InventoryDao
9 from dao.OrderDao import OrderDao
0 from dao.OrderDetailsDao import OrderDetailsDao
1 from dao.ProductDao import ProductDao
2
3
4 def main():
5     customer_obj = CustomerDao()
6     product_obj = ProductDao()
7     order_obj = OrderDao()
8     orderdetails_obj = OrderDetailsDao()
9     inventory_obj = InventoryDao()
0
1     # CRUD OPERATIONS
2     print(
3         "1. Customers\n2. Products\n3. Orders\n4. OrderDetails\n5. Inventory\n6. Exit\n"
4     )
5     choice = input("Enter your choice: ")
6

```

Operations on Customer:

```

26
27 if choice == "1":
28     print(
29         "\n1. Insert Customer\n2. Delete Customer\n3. Calculate Total Orders\n4. Get Customer Details\n5. Update Customer Info\n"
30     )
31     choice_customer = input("Enter your choice: ")
32     if choice_customer == "1":
33         first_name = input("Enter First Name: ")
34         last_name = input("Enter Last Name: ")
35         email = input("Enter Email: ")
36         phone = input("Enter Phone: ")
37         address = input("Enter Address: ")
38         customer_obj.insertCustomer(first_name, last_name, email, phone, address)
39     elif choice_customer == "2":
40         customer_id = input("Enter Customer ID to delete: ")
41         customer_obj.deleteCustomer(customer_id)
42     elif choice_customer == "3":
43         customer_id = input("Enter Customer ID: ")
44         customer_obj.calculateTotalOrders(customer_id)
45     elif choice_customer == "4":
46         customer_id = input("Enter Customer ID: ")
47         customer_obj.getCustomerDetails(customer_id)
48     elif choice_customer == "5":
49         customer_id = input("Enter Customer ID: ")
50         email = input("Enter New Email: ")
51         phone = input("Enter New Phone: ")
52         address = input("Enter New Address: ")
53         customer_obj.updateCustomerInfo(customer_id, email, phone, address)
54     else:
55         print("Invalid choice. Please try again.")
56

```

Operations on Product:

```
56         print("Invalid choice. Please try again.")
57     elif choice == "2":
58         print(
59             "\n1. Get Product Details\n2. Insert Product\n3. Update Product Info\n4. Check Product Stock\n"
60         )
61         choice_product = input("Enter your choice: ")
62         if choice_product == "1":
63             product_id = input("Enter Product ID: ")
64             product_obj.get_product_details(product_id)
65         elif choice_product == "2":
66             product_name = input("Enter Product Name: ")
67             description = input("Enter Product Description: ")
68             price = input("Enter Product Price: ")
69             product_obj.insert_product(product_name, description, price)
70         elif choice_product == "3":
71             product_id = input("Enter Product ID: ")
72             price = input("Enter New Price: ")
73             description = input("Enter New Description: ")
74             product_obj.update_product_info(product_id, price, description)
75         elif choice_product == "4":
76             product_id = input("Enter Product ID: ")
77             product_obj.is_product_in_stock(product_id)
78         else:
79             print("Invalid choice. Please try again.")
```

Operations On Order:

```
80
81     elif choice == "3":
82         print(
83             "\n1. Insert Order\n2. Update Order Status\n3. Cancel Order\n4. Calculate total amount\n5. Get order details"
84         )
85         choice_order = input("Enter your choice: ")
86         if choice_order == "1":
87             customer_id = input("Enter Customer ID: ")
88             total_amount = input("Enter Total Amount: ")
89             order_obj.insert_order(customer_id, total_amount)
90         elif choice_order == "2":
91             order_id = input("Enter Order ID: ")
92             new_status = input("Enter New Status: ")
93             order_obj.update_order_status(order_id, new_status)
94         elif choice_order == "3":
95             order_id = input("Enter Order ID: ")
96             order_obj.cancel_order(order_id)
97         elif choice_order == "4":
98             order_id = input("Enter Order ID: ")
99             order_obj.calculate_total_amount(order_id)
100        elif choice_order == "5":
101            order_id = input("Enter Order ID: ")
102            order_obj.get_order_details(order_id)
103        else:
104            print("Invalid choice. Please try again.")
```

Operations on OrderDetails:

```

05
06 elif choice == "4":
07     print(
08         "1. Calculate SubTotal\n"
09         + "2. Get Order Details\n"
10         + "3. Update Quantity\n"
11         + "4. Add Discount"
12     )
13     choice_order_detail = input("Enter your choice: ")
14     if choice_order_detail == "1":
15         order_detail_id = input("Enter Order Detail ID: ")
16         orderdetails_obj.calculate_subtotal(order_detail_id)
17     elif choice_order_detail == "2":
18         order_detail_id = input("Enter Order Detail ID: ")
19         orderdetails_obj.get_order_detail_info(order_detail_id)
20     elif choice_order_detail == "3":
21         order_detail_id = input("Enter Order Detail ID: ")
22         new_quantity = int(input("Enter new quantity: "))
23         orderdetails_obj.update_quantity(order_detail_id, new_quantity)
24     elif choice_order_detail == "4":
25         try:
26             order_detail_id = input("Enter Order Detail ID: ")
27             totalAfterDiscount = int(input("Enter Discount Percentage: "))
28             orderdetails_obj.add_discount(order_detail_id, totalAfterDiscount)
29         except ValueError:
30             print("Invalid input. Discount must be number.")
31     else:
32         print("Invalid choice. Please try again.")
33

```

Operations on Inventory:

```

132         print("Invalid choice. Please try again. ")
133
134 elif choice == "5":
135     print(
136         "1. Get Product\n"
137         + "2. Get Quantity in Stock\n"
138         + "3. Add To Inventory\n"
139         + "4. Remove From Inventory\n"
140         + "5. Update Stock Quantity\n"
141         + "6. Product Available\n"
142         + "7. Get Inventory Value\n"
143         + "8. List Low Stock Products\n"
144         + "9. List Out Of Stock Products\n"
145         + "10. List All Products"
146     )
147     choice_inventory = input("Enter your choice: ")
148     if choice_inventory == "1":
149         inventory_id = int(input("Enter the inventory id: "))
150         product = inventory_obj.get_product(inventory_id)
151         print(product)
152     elif choice_inventory == "2":
153         inventory_id = int(input("Enter the inventory id: "))
154         print(inventory_obj.get_quantity_in_stock(inventory_id))
155     elif choice_inventory == "3":
156         product_id = int(input("Enter the product id: "))
157         quantity_in_stock = int(input("Enter the quantity in stock: "))
158         last_stock_update = input("Enter the last stock update: ")
159         inventory_obj.insert_inventory(
160             product_id, quantity_in_stock, last_stock_update
161         )

```

```

162         elif choice_inventory == "4":
163             inventory_id = int(input("Enter the inventory id: "))
164             quantity = int(input("Enter the quantity: "))
165             inventory_obj.remove_from_inventory(quantity, inventory_id)
166         elif choice_inventory == "5":
167             inventory_id = int(input("Enter the inventory id: "))
168             new_quantity = int(input("Enter the new quantity: "))
169             inventory_obj.update_stock_quantity(inventory_id, new_quantity)

```

```

170         elif choice_inventory == "6":
171             inventory_id = input("Enter Inventory ID: ")
172             quantity_to_check = int(input("Enter Quantity to Check: "))
173             if inventory_obj.is_product_available(inventory_id, quantity_to_check):
174                 print("Product is available in inventory.")
175             else:
176                 print("Product is not available in inventory.")
177         elif choice_inventory == "7":
178             inventory_id = input("Enter Inventory ID: ")
179             print(inventory_obj.get_inventory_value(inventory_id))
180         elif choice_inventory == "8":
181             threshold = int(input("Enter the threshold for low stock: "))
182             inventory_obj.list_low_stock_products(threshold)
183         elif choice_inventory == "9":
184             inventory_obj.list_out_of_stock_products()
185         elif choice_inventory == "10":
186             inventory_obj.list_all_products()
187         else:
188             print("Invalid choice. Please try again.")

```

```

188             print("Invalid choice. Please try again.")
189         elif choice == '6':
190             print("Exiting...")
191         else:
192             print("Invalid choice. Please try again.")
193
194

```


Tasks Menu:

```
8
9  from dao.paymentDao import PaymentProcessingSystem
10 from exception.exceptions import IncompleteOrderException
11 from dao.CustomerDao import CustomerDao
12 from dao.InventoryDao import InventoryDao
13 from dao.OrderDao import OrderDao
14 from dao.OrderDetailsDao import OrderDetailsDao
15 from dao.ProductDao import ProductDao
16
17
18
19 class Menu:
20     def __init__(self):
21         self.customer_obj = CustomerDao()
22         self.product_obj = ProductDao()
23         self.order_obj = OrderDao()
24         self.orderdetails_obj = OrderDetailsDao()
25         self.inventory_obj = InventoryDao()
26         self.payment_obj = PaymentProcessingSystem()
27
28     def display_menu(self):
29         print("1. Customer Registration")
30         print("2. Product Catalog Management")
31         print("3. Place Orders")
32         print("4. Track Order Status")
33         print("5. Inventory Management")
34         print("6. Sales Reports")
35         print("7. Customer Account Updates")
36         print("8. Payment Processing")
37         print("9. Product Search and Recommendations")
38
39     def handle_choice(self, choice):
40         if choice == "1":
41             self.customer_registration()
42         elif choice == "2":
43             self.product_catalog_management()
44         elif choice == "3":
45             self.place_orders()
46         elif choice == "4":
47             self.track_order_status()
48         elif choice == "5":
49             self.inventory_management()
50         elif choice == "6":
51             self.sales_reports()
52         elif choice == "7":
53             self.customer_account_updates()
54         elif choice == "8":
55             self.payment_processing()
56         elif choice == "9":
57             self.product_search_and_recommendations()
58         else:
59             print("Invalid choice")
60
```

1: Customer Registration

Description: When a new customer registers on the TechShop website, their information (e.g., name, email, phone) needs to be stored in the database.

Task: Implement a registration form and database connectivity to insert new customer records. Ensure proper data validation and error handling for duplicate email addresses.

```
58
59     def customer_registration(self):
60         first_name = input("Enter first name: ")
61         last_name = input("Enter last name: ")
62         email = input("Enter email: ")
63         phone = input("Enter phone number: ")
64         address = input("Enter address: ")
65         self.customer_obj.insertCustomer(first_name, last_name, email, phone, address)
66
```

2: Product Catalog Management

Description: TechShop regularly updates its product catalog with new items and changes in product details (e.g., price, description). These changes need to be reflected in the database.

Task: Create an interface to manage the product catalog. Implement database connectivity to update product information. Handle changes in product details and ensure data consistency.

```
66
67     def product_catalog_management(self):
68         print(
69             "\n1. Get Product Details\n2. Insert Product\n3. Update Product Info\n4. Check Product Stock\n"
70         )
71         choice_product = input("Enter your choice: ")
72         if choice_product == "1":
73             product_id = input("Enter Product ID: ")
74             self.product_obj.get_product_details(product_id)
75         elif choice_product == "2":
76             product_name = input("Enter Product Name: ")
77             description = input("Enter Product Description: ")
78             price = input("Enter Product Price: ")
79             self.product_obj.insert_product(product_name, description, price)
80         elif choice_product == "3":
81             product_id = input("Enter Product ID: ")
82             price = input("Enter New Price: ")
83             description = input("Enter New Description: ")
84             self.product_obj.update_product_info(product_id, price, description)
85         elif choice_product == "4":
86             product_id = input("Enter Product ID: ")
87             self.product_obj.is_product_in_stock(product_id)
88         else:
89             print("Invalid choice. Please try again.")
90
```

3: Placing Customer Orders

Description: Customers browse the product catalog and place orders for products they want to purchase. The orders need to be stored in the database.

Task: Implement an order processing system. Use database connectivity to record customer orders, update product quantities in inventory, and calculate order totals.

```

91
92     def place_orders(self):
93         try:
94             product_id = int(input("Enter Product ID: "))
95             quantity = int(input("Enter quantity: "))
96             customer_id = int(input("Enter Customer ID: "))
97
98             if not self.inventory_obj.is_product_available(product_id, quantity):
99                 raise IncompleteOrderException("Selected product is not available in the required quantity.")
100
101             order_id = self.order_obj.insert_order(customer_id)
102             self.orderdetails_obj.insert_order_detail(order_id, product_id, quantity)
103             self.inventory_obj.remove_from_inventory(product_id, quantity)
104             self.order_obj.calculate_total_amount(order_id)
105             print("Order placed successfully.")
106         except IncompleteOrderException as e:
107             print("Error placing order:", e)
108         except Exception as ex:
109             print("Error placing order:", ex)

```

4: Tracking Order Status

Description: Customers and employees need to track the status of their orders. The order status information is stored in the database.

Task: Develop a feature that allows users to view the status of their orders. Implement database connectivity to retrieve and display order status information.

```

110
111
112     def track_order_status(self):
113         order_id = input("Enter order ID: ")
114         self.order_obj.get_order_details(order_id)
115

```

5: Inventory Management

Description: TechShop needs to manage product inventory, including adding new products, updating stock levels, and removing discontinued items.

Task: Create an inventory management system with database connectivity. Implement features for adding new products, updating quantities, and handling discontinued products.

```

115
116     def inventory_management(self):
117         print(
118             "1. Get Product\n"
119             + "2. Get Quantity in Stock\n"
120             + "3. Add To Inventory\n"
121             + "4. Remove From Inventory\n"
122             + "5. Update Stock Quantity\n"
123             + "6. Product Available\n"
124             + "7. Get Inventory Value\n"
125             + "8. List Low Stock Products\n"
126             + "9. List Out Of Stock Products\n"
127             + "10. List All Products"
128         )
129         choice_inventory = input("Enter your choice: ")
130         if choice_inventory == "1":
131             inventory_id = int(input("Enter the inventory id: "))
132             product = self.inventory_obj.get_product(inventory_id)
133             print(product)
134         elif choice_inventory == "2":
135             inventory_id = int(input("Enter the inventory id: "))
136             print(self.inventory_obj.get_quantity_in_stock(inventory_id))
137         elif choice_inventory == "3":
138             product_id = int(input("Enter the product id: "))
139             quantity_in_stock = int(input("Enter the quantity in stock: "))
140             last_stock_update = input("Enter the last stock update: ")
141             self.inventory_obj.insert_inventory(
142                 product_id, quantity_in_stock, last_stock_update
143             )
144         elif choice_inventory == "4":
145             product_id = int(input("Enter the product id: "))
146             quantity = int(input("Enter the quantity: "))
147             self.inventory_obj.remove_from_inventory(product_id, quantity)
148         elif choice_inventory == "5":
149             inventory_id = int(input("Enter the inventory id: "))
150             new_quantity = int(input("Enter the new quantity: "))
151             self.inventory_obj.update_stock_quantity(inventory_id, new_quantity)

```

```

152         elif choice_inventory == "6":
153             inventory_id = input("Enter Inventory ID: ")
154             quantity_to_check = int(input("Enter Quantity to Check: "))
155             if self.inventory_obj.is_product_available(inventory_id, quantity_to_check):
156                 print("Product is available in inventory.")
157             else:
158                 print("Product is not available in inventory.")
159         elif choice_inventory == "7":
160             inventory_id = input("Enter Inventory ID: ")
161             print(self.inventory_obj.get_inventory_value(inventory_id))
162         elif choice_inventory == "8":
163             threshold = int(input("Enter the threshold for low stock: "))
164             self.inventory_obj.list_low_stock_products(threshold)
165         elif choice_inventory == "9":
166             self.inventory_obj.list_out_of_stock_products()
167         elif choice_inventory == "10":
168             self.inventory_obj.list_all_products()
169         else:
170             print("Invalid choice. Please try again.")
171

```

6: Sales Reporting

Description: TechShop management requires sales reports for business analysis. The sales data is stored in the database.

Task: Design and implement a reporting system that retrieves sales data from the database and generates reports based on specified criteria.

```
104
105     def fetch_sales_data(self, start_date, end_date):
106         try:
107             connection = DBConnUtil.getConnection()
108             cursor = connection.cursor()
109             sql_query = """
110                 SELECT o.OrderID, o.OrderDate, o.TotalAmount, od.ProductID, od.Quantity
111                 FROM orders o
112                 INNER JOIN orderdetails od ON o.OrderID = od.OrderID
113                 WHERE o.OrderDate BETWEEN %s AND %s
114                 """
115             cursor.execute(sql_query, (start_date, end_date))
116             sales_data = cursor.fetchall()
117
118             # Fetch column names
119             column_names = [i[0] for i in cursor.description]
120
121             # Print column names
122             print("\t".join(column_names))
123
124             # Print data with column names
125             for row in sales_data:
126                 print("\t".join(str(val) for val in row))
127
128         except mysql.connector.Error as e:
129             print("Error fetching sales data:", e)
130         finally:
131             connection.close()
132
```

```
172     def sales_reports(self):
173         start_date = input("Enter Start Date: ")
174         end_date = input("Enter end date: ")
175         self.order_obj.fetch_sales_data(start_date=start_date, end_date=end_date)
176
```

7: Customer Account Updates

Description: Customers may need to update their account information, such as changing their email address or phone number.

Task: Implement a user profile management feature with database connectivity to allow customers to update their account details. Ensure data validation and integrity.

```
176
177     def customer_account_updates(self):
178         customer_id = input("Enter Customer ID: ")
179         email = input("Enter New Email: ")
180         phone = input("Enter New Phone: ")
181         address = input("Enter New Address: ")
182         self.customer_obj.updateCustomerInfo(customer_id, email, phone, address)
183
```

8: Payment Processing

Description: When customers make payments for their orders, the payment details (e.g., payment method, amount) must be recorded in the database.

Task: Develop a payment processing system that interacts with the database to record payment

transactions, validate payment information, and handle errors.

```
5 class PaymentProcessingSystem:
6
7     def record_payment(self, order_id, payment_method, amount):
8         try:
9             connection = DBConnUtil.getConnection()
10            cursor = connection.cursor()
11            sql_query = """
12                INSERT INTO payments (OrderID, PaymentMethod, Amount)
13                VALUES (%s, %s, %s)
14            """
15            cursor.execute(sql_query, (order_id, payment_method, amount))
16            connection.commit()
17
18            print("Payment recorded successfully")
19        except Error as e:
20            print("Error recording payment:", e)
21        finally:
22            connection.close()
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

9: Product Search and Recommendations

Description: Customers should be able to search for products based on various criteria (e.g., name, category) and receive product recommendations.

Task: Implement a product search and recommendation engine that uses database connectivity to retrieve relevant product information

```

92
93     def search_product_by_name(self, keyword):
94         try:
95             sql_query = """
96                 SELECT ProductID, ProductName, Description, Price
97                 FROM Products
98                 WHERE ProductName LIKE %s
99             """
100             self.cursor.execute(sql_query, ('%' + keyword + '%',))
101             products = self.cursor.fetchall()
102             if products:
103                 print("Search Results:")
104                 for product in products:
105                     print(product)
106             else:
107                 print("No products found matching the search criteria.")
108         except Exception as e:
109             print("Error searching for products:", e)
110

```

```

114
115     def get_product_recommendations(self, category):
116         try:
117             # Get product recommendations based on the specified category
118             connection = DBConnUtil.getConnection()
119             cursor = connection.cursor()
120             sql_query = """
121                 SELECT ProductID, ProductName, Description, Price
122                 FROM Products
123                 WHERE Category = %s
124                 LIMIT 5
125             """
126             cursor.execute(sql_query, (category,))
127             recommendations = cursor.fetchall()
128             if recommendations:
129                 print("Product Recommendations:")
130                 for product in recommendations:
131                     print(product)
132             else:
133                 print("No recommendations found for the specified category.")
134         except Exception as e:
135             print("Error getting product recommendations:", e)
136         finally:
137             connection.close()
138

```

```

92     def product_search_and_recommendations(self):
93         print("1. Search for a product\n2. Get Recommendations\n")
94         choice = input("Enter your choice: ")
95         if choice == '1':
96             keyword = input("Search for a product: ")
97             self.product_obj.search_product_by_name(keyword=keyword)
98         elif choice == '2':
99             category = input("Enter a category: ")
100            self.product_obj.get_product_recommendations(category=category)
101         else:
102             print("Invalid choice. Try again")

```

Menu Output:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop> & C:/Users/raaji/AppData/Local/Programs/Python/Python39-64/Python.exe hexaware training/TechShop/main/tasks.py
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 

```

Customer Registration:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop> & C:/Users/raaji/AppData/Local/Programs/Python/Python39-64/Python.exe hexaware training/TechShop/main/tasks.py
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 1
Enter first name: max
Enter last name: ver
Enter email: ver@mail.com
Enter phone number: 12345678
Enter address: monaco
Connected to MySQL database successfully.
Customer inserted successfully.

```

Product Catalog Management:


```
hexaware training/TechShop/main/tasks.py"ware training\TechShop>
```

```
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 2
```

```
1. Get Product Details
2. Insert Product
3. Update Product Info
4. Check Product Stock
```

```
Enter your choice: 1
Enter Product ID: 1
Connected to MySQL database successfully.
Product ID: 1
Product Name: HP Laptop
Description: High performance laptop
Price: 54999.99
```

Place Orders:

```
PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop> & C:/Users/raaji/AppData/Local/Programs/Python/Python38-32/Python.exe hexaware training/TechShop/main/tasks.py"
```

```
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 3
Enter Product ID: 3
Enter quantity: 1
Enter Customer ID: 20
Order inserted successfully.
Order detail inserted successfully.
Quantity removed from inventory successfully.
Total Amount calculated and updated successfully.
Order placed successfully.
```

```
& C:/Users/raaji/AppData/Local/Programs/Python/Python38-32/Python.exe
```

Track Order Status:

```
No details found for Order ID: 10 & C:/Users/raaji/AppData/Local/Programs/Python/Python38-32/Python.exe
```

```
hexaware training/TechShop/main/tasks.py"ware training\TechShop>
```

```
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 4
Enter order ID: 1
Order Details for Order ID: 1
Product: HP Laptop, Quantity: 1, Status: shipped
```

Inventory Management:

```
Enter your choice: 5
1. Get Product
2. Get Quantity in Stock
3. Add To Inventory
4. Remove From Inventory
5. Update Stock Quantity
6. Product Available
7. Get Inventory Value
8. List Low Stock Products
9. List Out Of Stock Products
10. List All Products
Enter your choice: 5
Enter the inventory id: 1
Enter the new quantity: 10
Stock quantity updated successfully.
```

Sales Report:

```
hexaware training/TechShop/main/tasks.py"ware training\TechShop>
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 6
Enter Start Date: 2024-01-01
Enter end date: 2024-12-12
OrderID OrderDate      TotalAmount      ProductID      Quantity
1      2024-04-01      54999.99      1      1
2      2024-04-02      329999.90      2      10
3      2024-04-03      16499.99      3      1
4      2024-04-04      10999.99      4      1
5      2024-04-05      8799.99 5      1
6      2024-04-06      38499.99      6      1
7      2024-04-07      60499.99      7      1
9      2024-04-09      10999.99      9      1
PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop>
```

Customer Account Updates:

```
hexaware training/TechShop/main/tasks.py"ware training\TechShop>
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 7
Enter Customer ID: 1
Enter New Email: alonso@mail.com
Enter New Phone: 12345678
Enter New Address: italy
Customer information updated successfully.
PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop>
```

Payment Processing:

```
PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop> cd C:\Users\raaji\AppData\Local\Programs\Python\Python39\Scripts
hexaware training/TechShop/main/tasks.py"
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 8
Enter Order ID: 1
Enter Payment Method: UPI
Enter amount: 54999
Payment recorded successfully
PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop>
```

Product Search:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 9
1. Search for a product
2. Get Recommendations

Enter your choice: 1
Search for a product: laptop
Search Results:
(1, 'HP Laptop', 'High performance laptop', Decimal('54999.99'))
(11, 'Dell Laptop', 'High performance laptop', Decimal('49999.99'))
PS C:\Users\raaji\OneDrive\Documents\hexaware training\TechShop>
```

Product Recommendations:

```
hexaware training/TechShop/main/tasks.py"ware training\TechShop>
1. Customer Registration
2. Product Catalog Management
3. Place Orders
4. Track Order Status
5. Inventory Management
6. Sales Reports
7. Customer Account Updates
8. Payment Processing
9. Product Search and Recommendations
Enter your choice: 9
1. Search for a product
2. Get Recommendations

Enter your choice: 2
Enter a category: mobile
Product Recommendations:
(2, 'Smartphone', 'Latest smartphone model', Decimal('32999.99'))
(12, 'mobile', 'gaming mobile', Decimal('90000.00'))
```